NXP : Cortex[™]-MO+ Hands-On Lab Using the LPCXpresso LPC800 *featuring MTB: Micro Trace Buffer* and ARM[®] Keil[™] MDK Toolkit Summer 2013 Version 2.1 by Robert Boys, <u>bob.boys@arm.com</u>

Hands-on using the NXP LPCXpresso LPC800 Board and Keil MDK

Introduction:

The latest version of this document is here: www.keil.com/appnotes/docs/apnt_237.asp

The purpose of this lab is to introduce you to the NXP LPC800 CortexTM-M0+ processor using the ARM[®] KeilTM MDK toolkit featuring the IDE μ Vision[®]. We will demonstrate all debugging features available on this processer including Micro Trace Buffer (MTB). At the end of this tutorial, you will be able to confidently work with these processors and Keil MDK. See www.keil.com/nxp for more labs, appnotes and other information. See <u>forums.arm.com</u>.

Keil MDK supports and has examples for most NXP ARM and 8051 processors. Check the Keil Device Database[®] on <u>www.keil.com/dd</u> for the complete list which is also included in MDK: In μ Vision, select Project/Select Device for target...

Linux: NXP ARM processors running Linux, Android and bare metal are supported by ARM DS-5[™]. www.arm.com/ds5.

Keil MDK-Lite[™] is a free evaluation version that limits code size to 32 Kbytes. Nearly all Keil examples will compile within this 32K limit. The addition of a valid license number will turn it into the appropriate commercial version.

Middleware: MDK Professional contains middleware libraries including TCP/IP stack, CAN drivers, a Flash file system and USB drivers. Contact Keil sales for information regarding middleware for your processor. <u>http://www.keil.com/arm/mdk.asp</u>.

RTX RTOS: RTX now comes with a Berkeley BSD type license. All variants of MDK contain the full version of RTX with Source Code. See <u>www.keil.com/rl-arm/kernel.asp</u> and <u>www.arm.com/cmsis</u>.

Why Use Keil MDK ?

MDK provides these features particularly suited for Cortex-M users:

- μVision IDE with Integrated Debugger, Flash programmer and the ARM[®] Compiler and assembler toolchain. MDK is a complete turn-key tool solution.
- 2. A full feature Keil RTOS called RTX is included with MDK. RTX comes with a BSD type license.
- 3. RTX Kernel Awareness windows. It is updated in real-time. All applicable ARM debugging technology is supported.
- 4. Available Debug Adapters: ULINK[™]2, ULINK-ME, ULINK*pro*, J-Link and NXP LPC-Link 2 (CMSIS-DAP).
- 5. Keil Technical Support is included for one year and is easily renewable. This helps your project get completed faster.
- **6.** MDK includes support for NXP processors including 8051, ARM7TM, ARM9TM, Cortex-M0, Cortex-M3, and Cortex-M4.

This document includes details on these features plus more:

- 1. Micro Trace Buffer (MTB). This is Instruction Trace. It is a history of where your program has been.
- 2. Real-time Read and Write to memory locations for Watch and Memory windows. These are non-intrusive to your program. No CPU cycles are stolen. No instrumentation code is added to your source files.
- 3. Four Hardware Breakpoints (can be set/unset on-the-fly) and two Watchpoints (also known as Access Breaks).
- 4. RTX and its RTX Tasks window: a kernel awareness program for RTX that updates while your program is running.
- 5. A DSP example program using ARM CMSIS-DSP libraries.

Micro Trace Buffer (MTB):

MDK supports MTB with ULINK2/ME, ULINK*pro* or LPC-Link 2 (CMSIS-DAP). MTB provides instruction trace which is essential for solving tricky program flow and other related problems. How to configure and use MTB is described.



Keil ULINK2 connected to LPCXpresso. The USB cable provides power to the LPC800.

Index:

	1.	NXP Evaluation Boards & Keil Evaluation Software:	3
	2.	Keil Software Installation:	3
	3.	CoreSight Definitions:	4
	4.	Debug Adapter Summary for Keil MDK with µVision IDE:	4
	5.	Debug Adapter Connection Issues with the LPCXpresso board:	5
	6.	CMSIS: Cortex Microcontroller Software Interface Standard:	5
	7.	Programming LPC-Link 2 as a CMSIS-DAP Complaint debug adapter:	6
Usin	g tł	e Blinky example with a ULINK2 or LINK-ME Debug Adapter:	
	8.	Configure µVision for various debug adapters:	7
	9.	Blinky example using the LPCXpresso LPC800:	8
	10.	Hardware Breakpoints:	8
	11.	Call Stack & Locals window:	9
	12.	Watch and Memory windows and how to use them:	10
	13.	Watchpoints: Conditional Breakpoints:	11
	14.	MTB: Micro Trace Buffer:	12
	15.	Exploring the MTB:	13
	16.	Configuring the MTB Micro Trace Buffer:	14
	17.	Trace "In the Weeds" Example:	15
	18.	RTX_Blinky: Keil RTX RTOS example:	16
	19.	RTX Kernel Awareness using RTX Viewer:	17
	20.	DSP Sine Example using ARM CMSIS-DSP Libraries	18
	21.	Configuring MTB for the DSP SINE example: (for reference)	20
Addi	itio	nal Information:	
	22.	Creating your own project from scratch:	20
	23.	LPC800 Cortex-M0+ Trace Summary:	23
	24.	Useful Documents:	23
	25.	Keil Products and contact information:	24

Using this document:

- 1. The latest version of this document and the necessary DSP example source files are available here: www.keil.com/appnotes/docs/apnt_237.asp
- You need a debug adapter: a ULINK2, ULINK-ME, ULINK*pro*, Segger J-Link (black case V6 or later) or a LPC-Link 2. MDK supports any of these with LPCXpresso LPC800. MTB is not currently supported with a J-Link.
- 3. ULINK-ME is used by default in this document. A ULINK-ME is electrically essentially the same as a ULINK2. Configuring another debug adapter such as a ULINK*pro* or a J-Link is easy to do.

4. Important note:

If you are using a debug adapter other than ULINK-ME: please see Debug Adapter Connection Issues on page 5.

5. **The first exercise starts on page 8.** You can go directly there if you are using ULINK2 or ULINK-ME. For any ULINK you must select the correct adapter in the Target selector box as shown there: It is easy to add these targets.

TIP: If you select ULINK or ULINK*pro*, and have the opposite ULINK actually connected to your PC; the error message will say "No ULINK device found". This message actually means that μ Vision found the wrong Keil adapter connected, and not that no ULINK was attached. Select the correct ULINK.

1) NXP Evaluation Boards & Keil Evaluation Software:

Keil makes many boards supporting NXP processors. Visit <u>www.keil.com/nxp</u> and select <u>ARM-based Device Families</u>. We have board support packages (BSP) for many other boards. Scan C:\Keil\ARM\Boards to find these example programs.

For NXP LPC3000 ARM9 processors using Linux, Android or bare metal (no OS) please see www.arm.com/ds5.

The directory \RL consists of middleware examples. Such middleware is a component of MDK Professional. To run these examples a full license is needed. Please contact Keil sales for a temporary license if you want to evaluate Keil middleware.

Other Keil Hands-On Labs for NXP Processors:

CAN www.keil.com/download/docs/385.asp

NGX XpLorer LPC4300 www.keil.com/appnotes/docs/apnt 233.asp

Keil MCB1700 - LPC1700 www.keil.com/download/docs/394.asp

2) Keil Software Installation:

This document was written for Keil MDK 4.70 or later which contains μ Vision 4. The evaluation copy of MDK (MDK-Lite) is available free on the Keil website. Do not confuse μ Vision4 with MDK 4.0. The number "4" is a coincidence.

MDK 4.71a is the current official Keil release. (June 2013)

To obtain a copy of MDK go to www.keil.com/arm and select the "Download" icon located on the right side.

Install MDK into the default directory. You can install MDK into any directory, but references in this lab are made to C:\Keil.

MDK-ARM_AddOn_LPC8xx.zip: Do **not** install this file from the NXP website. All the files needed to debug on LPCXpresso LPC800 are included on MDK 4.71. Get the DSP example from: <u>www.keil.com/appnotes/docs/apnt_237.asp</u>

MDK 4.71 has:

- Complete support for MTB trace.
- MTB Trace works with ULINK2, ULINK-ME (the version with 10 pin connector) and ULINK*pro*. Currently, Segger J-Link (black case) does not work with MTB. However, everything else works with it.
- All the example files. For the DSP example go here: www.keil.com/appnotes/docs/apnt 237.asp.

You can use the evaluation version of MDK-Lite and ULINK2, ULINK-ME, ULINK*pro* for this lab. You can use a J-Link (black case) for everything except MTB. This will likely come at a later date.

		Disassembly	Ф X
	[0x0000015E 6008 STR r0,[r1,#0x00] 🔽 🗖
Watch 1	Ф X	32: LED_Out (0);	<pre>: 0; i < 10; i++)NOP();</pre>
Name Value	Туре	0x00000162 F7FFFD6 BL.W LED_Out (0x0	0000112)
···· 🔶 leds 0x0000001	unsigned int	33:) 0x00000166 BD00 POP (pc)	B 0x0000624
	unsigned short	0x00000168 0688 DCW 0x0688	NOP
<enter expression=""></enter>	-	0x0000016A 0000 DCW 0x0000	ADDS r3,r3,#1
	1.	0x0000016E A000 DCW 0xA000	CMP r3,#0x0A
		Memory 1	ф х 🕨
LED.c Blinky.c Dbg_MTB.ini	startup_LPC8xx.s	Jction Trace	
136 HardFault_Handler\		Address: 0	
137 PROC	Watch 1		
a 139 B	ault_Hand	[00000000 0000000
140 ENDP	Name	Value Type	EFFFFB61 0000000
141 SVC Handler PROC	🛛 🖓 Ied	0x0000002 unsigned	d int 00000000 00000000 1;
57: leds <<= 1;	🔷 diy	/al 0x01E0 unsigned	d short 00000005 0000005
>x000001E6_006D	Entor	overaction>	
Call Stack + Locals		4 ×	nory 1
Name Location/Value	Туре	als Watch 1 Memory 1	
↓ ⊟	int f0	ais watch I memory I	
✓ leds 0x0000004	auto - unsigned in		Saura
✓ db//al 0x0020	auto - unsigned sh	ort 6 D3EB BCC 0x00000620	
	auto - unsigneu si	0 BF00 NOP	LED.c <u>Blinky.c</u> CMP.c <u>startup</u>
Call Stack + Locals Memory 1		P.	39 - /*
55 dlyVal = dlyVal << 5;	2002	DCW 0x0688	40 Main Program
56	3062	KUUUUU16A UUUU DCW 0x0000	42 Fint main (void) {
▶ 57 leds <<= 1;	3064	x0000016E A000 DCW 0xA000	<pre>43 static uint32 t leds = 0x01;</pre>
58 if (leds >= (1 << LED_NUM)) 1	Leds = 0 3065	k000000DC E7FE B HardFault Handler (0x00000	ODC) 44 static uint16 t dlyVal = 0;
59 LED_Out (leds);	3066	x000000DC E7FE B HardFault_Handler (0x00000	0DC) 45
ou Deray(dryVal);	1		
	Disassembly	Instruction Trace	

3

3) CoreSight Definitions: It is useful to have a basic understanding of these terms:

Note: The Cortex-M0+ options are highlighted in red below: NXP Cortex-M3 and Cortex-M4 have all except MTB. Consult your specific device datasheet for specifics.

- JTAG: Provides access to the CoreSight debugging module located on the Cortex processor. It uses 4 to 5 pins.
- SWD: Serial Wire Debug is a two pin alternative to JTAG and has about the same capabilities except for no Boundary Scan. SWD is referenced as SW in the μVision Cortex-M Target Driver Setup. See page 5, 1st screen. The SWJ box must be selected. The LPC800 processors use SWD exclusively. There is no JTAG on the LPC800.
- SWV: Serial Wire Viewer: A trace capability providing display of reads, writes, exceptions, PC Samples and printf. SWV must use SWD because of the TDIO conflict described in **SWO** below.
- DAP: Debug Access Port. A component of the ARM CoreSight debugging module that is accessed via the JTAG or SWD port. One of the features of the DAP are the memory read and write accesses which provide on-the-fly memory accesses without the need for processor core intervention. µVision uses the DAP to update memory, Watch and a RTOS kernel awareness window in real-time while the processor is running. You can also modify variable values on the fly. No CPU cycles are used, the program can be running and no code stubs are needed in your sources. You do not need to configure or activate DAP. µVision does this automatically when you select the function.
- SWO: Serial Wire Output: SWV frames usually come out this one pin output. It shares the JTAG signal TDIO.
- Trace Port: A 4 bit port that ULINKpro uses to collect ETM frames and optionally SWV (rather than the SWO pin).
- ETM: Embedded Trace Macrocell: Provides all the executed instructions. Only ULINKpro provides ETM.
- ETB: Embedded Trace Buffer: a small dedicated on chip RAM (4 to 4 Kbytes) accessible with a debug adapter.
- MTB: Micro Trace Buffer. A portion of the device internal RAM is used for an instruction trace buffer. MTB is only on LPC800 Cortex-M0+ processors. Most NXP Cortex-M3 and Cortex-M4 processors provide ETM trace.

4) Debug Adapter Summary for Keil MDK with µVision IDE:

ULINK2: Pictured on page 1. This is a hardware JTAG/SWD debugger. It connects to various connectors found on boards populated with ARM processors. With NXP Cortex-M3 and M4 processors, ULINK2 adds Serial Wire Viewer (SWV). The LPC1700 lab <u>www.keil.com/download/docs/394.asp</u> has many SWV and ETM trace examples.

ULINK-ME: Pictured on page 8. ULINK-ME is provided only combined with a board package from Keil or an OEM. Electrically and functionally it is very similar to a ULINK2. With Keil µVision, they are used as equivalents. To connect to the LPCXpresso LPC800, you need the newer ULINK-ME with the 10 pin Cortex JTAG/SWD connector as shown on page 8.

ULINK*pro*: Pictured on the next page. ULINK*pro* is Keil's most advanced debug adapter. With NXP Cortex-M3 and Cortex-M4 processors, ULINK*pro* provides Serial Wire Viewer (SWV) and adds ETM instruction trace. Code Coverage, Performance Analysis and Execution Profiling are then provided using ETM. ULINK*pro* programs the flash very fast.

Segger J-Link: Pictured on the next page with adapter. µVision supports J-link and J-Link Ultra (black case only) Version 6.0 or later. The J-Link family supports all CoreSight components except MTB. This will be supported in the future.

LPC-Link 2: A CMSIS-DAP compliant debug adapter. This provides either a J-Link Lite or a CMSIS-DAP connection. μ Vision supports both. CMSIS-DAP has MTB trace working. J-Link is under development.

CMSIS-DAP is a new ARM standard where a small processor located on the target board acts as the Debug Adapter. It connects to your PC with a standard USB cable.

CMSIS-DAP provides run control debugging, Flash and RAM programming, Watchpoints, hardware breakpoints, and DAP reads and writes in Watch and memory windows updated in real-time as well as MTB trace and RTX System kernel awareness. You are able to easily incorporate a CMSIS-DAP design on your own custom target boards. For documents go to silver.arm.com/browse/CMSISDAP and download CMSIS-DAP Beta 0.01. Also see www.arm.com/cmsis.

NEW ! ULINK2 and ULINK-ME with CMSIS-DAP Support: (not in 4.71a but will be in 4.72)

Starting with MDK 4.70, you can use either of these as a CMSIS-DAP compliant debugger or in standard ULINK2/ME mode. CMSIS-DAP mode adds trace search, save and clear window options. This is the same window as with the ULINK*pro*.

The ULINK2/ME firmware needs to be updated.	This is automatic when you enter	Debug mode. 🔍 After this, you can
	Linker Debug Utilities	Linker Debug Utilities
select either mode in the Target Options menus:	Use: CMSIS-DAP Debugger Settings	Or Use: ULINK2/ME Cortex Debugger Settings

5) Debug Adapter Connection Issues with the LPCXpresso board Rev A:

These issues are very minor. Complete and easy workarounds are described:

ULINK-ME: No adjustments are needed. ULINK-ME ignores the missing VTRef voltage.

ULINK2: Pin 1 of the Debug connector (VTRef) is used by ULINK2 to determine the operating voltage of the target. The LPCXpresso by default leaves this pin floating so it has zero volts. Either populate R29 with a low ohms resistor or set the ULINK2 internal VCC jumper to 3.3V. This jumper is shown set correctly in the picture here:

ULINK*pro*: No adjustments are needed. ULINKpro Remove the cover to install the 20 to 10 pin cable. Do not remove or disturb the battery !

LPC-Link 2: No adjustments are needed. LPC-Link 2 ignores the VTRef voltage.

Segger J-Link: The J-Link pulls pin 10 of the debug connector (nSRST) down to about 1.5 volts and this holds the processor in RESET. This appears to be caused by VTRef floating instead of at 3.3 volts. Populate R29 (a glob of solder will do). Adding a jumper from J4 pin 11 to J6 pin 2 (+3.3v) to bring nSRST up to 3.3v prevents any debug adapter, including J-Link, to use this pin to reset the processor. By default, MDK does not use nSRST to reset the target. It uses the CPU SYSRESETREQ bit.



Go to www.keil.com and search for schematics for Keil ULINK interface details.





Segger J-Link Ultra with optional Cortex adapter.

USB on LPCXpresso board: If you plug the USB into the LPCXpresso board and Windows attempts to install USB drivers, please cancel this operation. We do not need any USB drivers for this board for this tutorial.

6) CMSIS: Cortex Microcontroller Software Interface Standard

CMSIS-CORE provides a common interface to ARM processor cores and peripheral registers. NXP example software is CMSIS hardware abstraction layer compliant.

CMSIS-DSP DSP libraries are currently offered for all Cortex-M0, Cortex-M0+, Cortex-M3 and Cortex-M4 processors.

CMSIS-RTOS provides standard APIs for RTOSs. RTX is a free RTOS available from ARM as part of CMSIS Version 3.0. RTX is included, with source files, with MDK and is also available here: www.keil.com/demo/eval/rtx.htm.

CMSIS-DAP (Debug Access Port) is a new ARM standard for on-board debugging adapters. An external adapter such as a ULINK2 is not needed for most debugging applications. LPC-Link 2 is CMSIS-DAP compliant. ULINK-ME and ULINK-2 have CMSIS-DAP operating modes and well as their regular proprietary option.

CMSIS-SVD (System View Description) are XML files that contain the programmer's view of a complete microcontroller system including peripherals. MDK supports CMSIS-SVD.

See www.arm.com/cmsis and forums.arm.com for more information.

7) Programming LPC-Link 2 as a CMSIS-DAP Compliant debug adapter:

LPC-Link 2 provides either a Segger J-Link2 Lite or CMSIS-DAP mode. μ Vision supports both. CMSIS-DAP mode provides MTB support. All files needed are here: <u>http://www.lpcware.com/lpclink2-config-tool</u>. **There are three steps to configure LPC-Link 2 with \muVision:**

- 1. Install the NXP USB drivers. The filename currently is <u>lpc18xx_43xx_winusb_drivers_v1.00.zip</u>. Extract these into a temp directory and plug the LPC-Link 2 into the PC USB and install the USB drivers in the usual way.
- 2. Your PC must have Microsoft .NET Framework 4 or later. When you run the LPC configuration tool in Step 3 and you do not have the correct version of .NET, there will be an error. If you do not receive an error, then your .NET is correct. You can get .NET here: www.microsoft.com/net/download.
- 3. Program the LPC-Link to either J-Link or CMSIS-DAP mode.

A program called LPC-Link 2 Configuration Tool is used. Obtain it from the website listed above: LPC-Link 2 Config tool.zip Extract it

into a suitable directory on your PC. You will execute it by double-clicking on it or in the fashion you prefer.

Configuring LPC-Link 2 for CMSIS-DAP:

- 1. Remove jumper JP1. Whether JP1 installed or not will be sensed by the processor at its startup routines.
- 2. Connect LPC-Link 2 to your PC's USB port. The USB drivers in Step 1 above must be installed and must connect.
- 3. Start LPC-Link 2 Config.exe by double-clicking on it. This file runs alone, it does not install as most programs do. The window below opens up. Follow the steps it is fairly simple and reliable.

TIP: JP1 must be either installed or not when the power is applied or cycled. In Step 1 below it is assumed the LPC-Link 2 is not connected to the USB port when you are instructed to remove JP1. If it is powered, just cycle the power with JP1 off.

TIP: If the instructions in Step 2 below are greyed out, this usually means the LPC-Link 2 is not connected to the PC USB drivers. You will have to fix this before you can continue. Solutions include: check the USB port or re-install the drivers.

TIP: Step 3 assumes power will be applied via the USB cable. After replacing JP1, the USB cable must be plugged in.

Configuring µVision for LPC-Link 2:

In μ Vision, Select Target Options and select the Debug tab. In the drop-down menu box select CMSIS-DAP Debugger as shown at the bottom of this page: You will also have to select the Flash programming algorithm in the Utilities tab.

You only need to program the	File Help		
LPC-Link 2 once. The firmware will stay in flash until you need to change it to a different mode.	Step 1 1: Select a Link2 program image [MSIS-DAP debugger mage CMSIS-DAP UPC-Link 2 debugger	Step 2 1: Press the program button to program the Link2 board	Step 3 1. Insert a jumper JP1 on the Link2 2. Disconnect the USB cable from the Link2 The board is now ready for operation!
You can now complete the instructions on the next page to configure uVision for the debug adapter you are using. TIP: JP2 supplies power to your target board if installed. You can leave JP2 on or off for all the programming steps above.	This image allows the Link2 board to operate as a CNSIS- DAP debugger that works with tool chains that support the CMSIS-DAP protocol such as Kell uVision, and IAR EWARM. ☐ Auto program currently selected Link2 image on board detection ☑ Enable nu-time debug messages 2: Remove jumper JPI on the Link2 3: Connect a USB cable between the Link2 and PC ☐ USB cable between the Link2 and PC	LPC-Link 2 C	Configuration program:
Selecting CMSIS-I	DAP: Uinker Debug Utilities CMSIS-DAP Debugge	r Settings	.i.



LPC-Link 2 connected to LPC800.

8) Configuring µVision for various debug adapters:

- 1. The LPC800 examples are pre-configured to use the ULINK2 or ULINK-ME. If you are using one of these adapters, you do not need to configure anything on this page. These steps show you how to change to other debug adapters such as ULINK*pro* or a J-Link and confirms you have successfully connected to the Cortex-M0++ core in Step 9.
- 2. Start μVision . Connect the LPCXpresso board to a ULINK2 or ULINK-ME or other supported debugger.
- 3. Power LPCXpresso with a USB cable. We will not use this USB connection for anything else. If a request from Windows to install USB drivers pops up: cancel this operation.
- 4. If µVision says a Firmware Upgrade is required to the ULINK, please allow this. It will not take very long.
- 5. Select Project/Open Project and select the project you want to use for your board. We used C:\Keil\ARM\Boards\Embedded Artists\LPCXpresso LPC800\Blinky\Blink.uvproj.
- 6. Select Target Options 🔊 or ALT-F7 and select the Debug tab. In the drop-down menu box select ULINK2/ME

Cortex Debugger or the adapter you want to use as shown here:

- 7. Select Settings and the next window below opens up. This is the control panel for debug control.
- In Port: select SW. Select the SWJ box. Note: The LPC800 has only SWD (SW). It does not have JTAG. You can do everything with SWD.
- 9. In the SW Device area: ARM CoreSight SW-DP MUST be displayed. This confirms you are connected to the target processor. If there is an error displayed or it is blank: this must be fixed before you can continue. Check the target power supply. Cycle the ULINK and LPC800 power.

bug Trace Flash Download	1			
ULINK USB - JTAG/SW Adapt	er SW De	vice		
Serial No: M2093ADE	-	IDCODE	Device Name	Move
ULINK Version: ULINK-ME Device Family: Cortex-M	SWDIO	⊙ 0x0BC11477	ARM CoreSight SW-DP	Up Down
Firmware Version: V2.01	© Aut	omatic Detection	ID CODE:	_
SWJ Port: SW 💌	С Ма	nual Configuration	Device Name:	
Max Clock: 1MHz	- Add	Delete U	odate	AP: 0x00

TIP: To refresh this screen, change the option in Port: to JTAG and then back to SW or click OK to leave and then re-enter.

10. Click OK once to return to the Options for target menu.

Configure the Keil Flash Programmer:

- 1. Select the Utilities tab.
- 2. Select ULINK2/ME Cortex Debugger or your own adapter as shown:
- 3. Click Settings to select the programming algorithm.
- 4. Select Add and select the LPC8xx Flash if necessary as shown here:
- LPC8xx IAP 16kB Flash is the correct one to use with the LPCXpresso LPC800 board.
- 6. Click on OK once.

TIP: To program the Flash every time you enter Debug mode, check Update Target before Debugging.

 Click on OK once more to return to the μVision main screen.

TIP: To use a ULINK*pro* or a J-Link, follow the instructions above but select the correct adapter. LPC-Link 2 will work with μ Vision. Make sure you check the debug connection issues on page 5.

col raiger output bann	g User C/C++ Asm	Linker De	bug Utilities
onfigure Flash Menu Comman	nd		
Use Target Driver for Flash	h Programming		
ULINK2/ME C	ortex Debugger	Settings	Update Target before Debugging
Init File:			Edit
war i no.]			
-M Target Driver Setup			
g Trace Flash Download	1		
and the second	10-10		
and Franking			and a state of the second
ownload Function		RAM for Al	gorithm
winload Function	Program	Start: 0x	10000000 Size: 0x0800
Diversion C Erase Full Chi C Erase Full Chi C Erase Sectors C Do not Erase	Program ✓ Program ✓ Verify ✓ Reset and Run	RAM for Al	gonthm 10000000 Size: 0x0800
winload Function C Erase Full Chi Frase Full Chi Frase Sectors Do not Erase ogramming Algorithm	Program Verify ▼ Reset and Run	Start: 0x	size: 0x0800
ownload Function C Erase Full Chi, C Erase Full Chi, C Erase Sectors C Do not Erase ogramming Algorithm Description	P V Program V Venfy Reset and Run	RAM for Al	10000000 Size: 0x0800
ownload Function Erase Full Chi Frase Full Chi Frase Sectors Do not Erase ogramming Agorithm Description PC3x IAP 16kB Flash	P Program Verfy Reset and Run Device Type	RAM for All	gorithm 10000000 Size: 0x0800 Address Range 00000000H - 00003FFFH 00003FFFH
overload Function C Erase Full Chi C Erase Sectors C Do not Erase ogramming Algorithm Description LPCBox IAP 16kB Rash		Carl Carl Carl Carl Carl Carl Carl Carl	30000000 Size: 0x0800 Address Range 00000000H - 00003FFFH
ownload Function C Frase Full Chi C Frase Sectors C Do not Erase ogramming Agorithm Description LPC8xx IAP 16kB Flash	D I Program I Verify I Reset and Run Device Type 1 On-chip Flash	Device Size	30000000 Size: 0x0800 Address Range 00000000H - 00003FFFH 00003FFFH
overload Function C Erase Full Chi C Erase Sectors C Do not Erase ogramming Algorithm Description LPC8ox IAP 16kB Flash		RAM for Al Start: 0x Device Size	30000000 Size: 0x0800 Address Range 00000000H - 00003FFFH 00003FFFH
overload Function C Erase Full Chi C Erase Sectors C Do not Erase ogramming Algorithm Description LPCBox IAP 16kB Rash	Program Venfy Venfy Venfy Proset and Run Device Type 1 On-chip Flash	RAM for Al Start: 0x Device Size	J0000000 Size: 0x0800 Address Range
overload Function C Erase Full Chi C Erase Sectors C Do not Erase ogramming Algorithm Description LPCbox IAP 16kB Flash	P I Program I Venfy I Reset and Run Device Type 1 On-chip Flash	RAM for Al Start: 0x Device Size 1 16k Start: 0x	J0000000 Size: 0x0800 Address Range

To use a LPC-Link 2, select CMSIS-DAP as the debug adapter in the Target Options window under the Debug tab.

9) Blinky example program using the NXP LPXpresso LPC800:

Now we will connect a Keil MDK system using the LPCXpress LPC800 and a ULINK2 or ME as the debug adapter.

- 11. Connect the ULINK as shown below or on page 1. Take care in attaching the debug cable. It is easy to get it backwards. The USB cable to J3 is used to only power the board. Cancel any requests to install USB drivers.
- 1. Start μ Vision by clicking on its desktop icon.
- 2. Select Project/Open Project.
- 3. Open the file: C:\Keil\ARM\Boards\Embedded Artists\LPCXpresso LPC800\Blinky\Blinky.uvproj.
- Select "LPC812M101 Flash" in the Select Target menu.
 LPC812M101 Flash
 This is where you create and select different target configurations such as to execute a program in RAM or Flash. This Target selection is pre-configured to use a ULINK2 or a ULINK-ME.
- 5. Compile the source files by clicking on the Rebuild icon. You can also use the Build icon beside it.
- Program the LPC800 flash by clicking on the Load icon: Progress is indicated in the Output Window.
- Enter Debug mode by clicking on the Debug icon. Select OK if the Evaluation Mode box appears.
 Note: You only need to use the Load icon to download to FLASH and not for RAM operation if it is chosen.
- 8. Click on the RUN icon. Note: you stop the program with the STOP icon.

The three colour LED D3 on the LPCXpresso board will now blink in sequence.

The speed the leds change can be changed by varying the pot R38.

Now you know how to compile a program, program it into the LPC800 processor Flash, run it and stop it !

Note: The board will start Blinky stand-alone. Blinky is now permanently programmed in the Flash (until reprogrammed). If you remove the ULINK and cycle the board power, this program will run automatically from the Flash memory.

10) Hardware Breakpoints:

The LPC800 has four hardware breakpoints that can be set or unset while the program is running.

- 1. With Blinky running, in the source window, set a hardware breakpoint by clicking on a darker block in the left in Blinky.c in the inside the while(1) loop.
- 2. A red circle will appear and the program will presently stop at this point.
- 3. Note the breakpoint is displayed in both the disassembly and source windows as shown here:
- 4. Every time you click on the RUN icon the program will run until the breakpoint is again encountered and the next LED will illuminate.
- 5. Remove the breakpoint by clicking on it.



Clicking in the C source window will indicate the appropriate code line in the Disassembly window and vice versa. Scrolling up and down in one window with the arrow keys moves the other window. This is relationship is indicated by the cyan arrow as shown at Line 57. The yellow arrow is the program counter.

TIP: A hardware breakpoint does **not** execute the instruction it lands on. ARM CoreSight hardware breakpoints are no-skid. This is a rather important feature.



8

11) Call Stack + Locals Window:

Local Variables:

The Call Stack and Locals windows are incorporated into one integrated window. Whenever the program is stopped, the Call Stack + Locals window will display call stack contents as well as any local variables belonging to the active function.

If possible, the values of the local variables will be displayed and if not, the message <not in scope> will be displayed. The Call + Stack window presence or visibility can be toggled by selecting View/Call Stack Window in the main μ Vision window when in Debug mode.

- 1. Set a breakpoint in the while(1) loop in main(). Run Blinky. It will soon stop on this breakpoint.
- 2. Click on the Call Stack + Locals tab if necessary to open it.
- 3. Shown is the Call Stack + Locals window.
- 4. The contents of the local variables are displayed as well as function names.
- In this example, two local variables leds and dlyval are displayed in the window here with their values:

Call Stack + Locals				
Name	Location/Value	Туре		
🖃 🔶 main	0x00000198	int f()		
- 🔗 leds	0x0000004	auto - unsigned int		
🛶 🧳 diyVal	0x0020	auto - unsigned short		
		·		
Call Stack + Locals	Memory 1			

- 6. Click on the Step icon or F11: ⁽¹⁾ Make sure the Blinky.c window is in focus else you will step in assembly.
- 7. Continue until the program enters the LED_Out function. The Call Stack + Locals window will now show this event:
- 8. Keep stepping until LED_On is entered as shown here:
- **9.** Click on the Step Out icon or CTRL-F11 once or twice to exit all function(s) to return to main(). This will be indicated in the Call Stack window.
- 10. When you are ready to continue, remove the hardware breakpoint by clicking on its red circle ! You can also type Ctrl-B and select Kill All.

Call Stack + Locals 📮			
Name	Location/Value	Туре	
🖃 🔶 LED_On	0x00000100	void f(unsigned int)	
🛶 num	0x0000001	param - unsigned int	
🖃 🔶 LED_Out	0x00000112	void f(unsigned int)	
🛶 value	0x0000002	param - unsigned int	
🔷 i	0x0000001	auto - int	
🗄 🔶 main	0x00000198	int f()	
🔷 🔗 leds	0x0000002	auto - unsigned int	
🔷 🔷 dlyVal	0x0020	auto - unsigned short	
		•	
Call Stack + Locals	Memory 1		

TIP: You can modify a variable value in the Call Stack & Locals window when the program is stopped.

TIP: This is standard "Stop and Go" debugging. ARM CoreSight debugging technology can do much better than this. You can display global or static variables updated in real-time while the program is running. No additions or changes to your code are required. Variable update while the program is running is not possible with local variables because they are usually stored in a CPU register. They must be converted to global or static variables so they always remain in scope: usually in RAM.

Call Stack:

The list of stacked functions is displayed when the program is stopped as you have seen. This is useful when you need to know which functions have been called and are stored on the stack. This can be quite useful for finding program crashes. Using MTB Micro Trace Buffer in conjunction with the Call Stack provides much more debugging power.

! Do not forget to remove the hardware breakpoint(s) before continuing.

TIP: You can access the Hardware Breakpoint table by clicking on Debug/Breakpoints or Ctrl-B. This is also where Watchpoints (also called Access Points) are configured. You can temporarily disable entries in this table by unchecking them. You can permanently deleted them by using one of the Kill buttons.

12) Watch and Memory Windows and how to use them:

The Watch and Memory windows will display updated variable values in real-time. It does this using the ARM CoreSight debugging technology that is part of Cortex-M processors. It is also possible to "put" or insert values into the Memory window in real-time. It is possible to "drag and drop" variable names into windows or enter them manually. You can also right click on a variable and select Add *varname* to... and select the appropriate window where you want it inserted.

Watch window:

Make a STATIC variable: Recall the Watch and Memory windows can't see local variables unless stopped in their function. If you have trouble using STATIC in the Watch and Memory windows,

make these variables global. Often, global variables are easier to use.

- Stop the processor \bigotimes and exit debug mode. 1.
- Add the static keyword to the variables **leds** and **dlyVal**. 2.
- Click on Rebuild i and program the Flash with Load 3
- Enter Debug mode. Q Click on RUN 4.
- You can configure a Watch window while the program is running. You can also do this with a Memory window. 5.
- Right-click on leds and select Add 'leds' to... and choose Watch 1. Repeat with dlyval. 6.
- The Watch 1 window below will display these variables. If the program is running, **leds** will change value as it 7. points to which LED is lit. The variable **dlyVal** will change as you vary the pot R38. This updating is in real time.
- Periodic Window Update 8. If these variables do not update: Select View and select Periodic Window Update:

TIP: You can also enter a variable by blocking it, click and hold it and drag it into a Watch or Memory window and then release it.

You can also enter a variable manually by double-clicking <Enter expression> or press F2 and use copy and paste or typing the variable name.

TIP: To Drag 'n Drop into a tab that is not active, block the variable, click it and hold the mouse over the tab you want to

open; when it opens, move your mouse in the window and release the variable.

Memory window:

- Make sure the program is running. 1.
- Right-click on leds and select Add leds to... and choose Memory 1. 2.
- 3. If the Memory window is not visible, this action will open it.
- Right click in the memory window and select Unsigned/Int. 4.
- Note the value of leds is displaying its address in Memory 1 as if it is a pointer. This is useful to see what address a 5 pointer is pointing to but this not what we want to see at this time.
- Add an ampersand "&" in front of the variable name and press Enter. The physical address is shown (0x1000 0008). 6.

Memory 1

Address: &leds

- 7. The data contents of **leds** is displayed as shown here:
- 8. Both the Watch and Memory windows are updated in real-time.
- 9. Right-click with the mouse cursor over the desired data field and select Modify Memory. You can change a memory or variable on-the-fly while the program is still running.

TIP: To view variables and their location use the Symbol window. Select View/Symbol Window while in Debug mode.

These Read and Write accesses are handled by the Serial Wire Debug (SWD) Port (DAP), which provides on-the-fly memory accesses. No CPU cycles are used to perform these operations.

0x10000014:	00000000	00000000	00000000
0x10000020:	00000000	00000000	00000000
0x1000002C:	00000000	00000000	0000000
Call Stack + L	ocals Watch	1 Memor	v 1
And and a second of a			·

0x10000008: 0000004 000001E0 0000000

	•					
Watch 1		ų ×				
Name	Value	Туре				
eds 🥐 leds	0x0000002	unsigned int				
🔗 dlyVal	0x01E0	unsigned short				
Enter expression>						
· · · · · · · · · · · · · · · · · · ·						
Call Stack + Locals	Watch 1 Memory 1					

📩 LED.c

39

40

41

43

44

45

42 🖃

Blinky.c

Main Program

int main (void) {

CMP.c

static uint32_t leds = 0x01;

static uint16 t dlyVal = 0;

📩 startup

×

۰

Copyright © 2013 ARM Ltd. All rights reserved www.keil.com

13) Watchpoints: Conditional Breakpoints

The LPC800 has two Watchpoints. Watchpoints can be thought of as conditional breakpoints. Watchpoints are also referred to as Access Breaks in Keil literature. Cortex-M0+ Watchpoints, unlike those in Cortex-M3 and Cortex-M4, are intrusive. The test for the condition in a Cortex-M0+ requires briefly halting the processor. It is restarted if the test is not true.

- 1. Use the same Blinky configuration as the previous page. Stop the program if necessary. Stay in debug mode.
- 2. We will use the static variable dlyval you created in Blinky.c to explore Watchpoints.
- 3. Select Debug in the main µVision window and then select Breakpoints or press Ctrl-B.
- 4. In the Expression box enter: "\\Blinky\Blinky.c\main\dlyVal==0x320" without the quotes.

TIP: You must fully qualify this variable because it is visible only when uVision is in the while(1) loop in main().

If **dlyVal** had been changed to a global variable, you normally do not need to fully qualify it.

5. Select both the Read and Write Access.

- 6. Click on Define and it will be accepted as shown here:
- 7. Click on Close.
- 8. Enter the variable **dlyval** to the Watch 1 window by dragging and dropping it if it is not already there.
- 9. Click on RUN.
- 10. You will see **dlyval** change in the Watch window.
- 11. Rotate the pot and when **dlyval** equals 0x320, the Watchpoint will stop the program. This can be tricky as this variable is not updated often.

☑ 00: (A readwrite 0x1000000C len=2), "\\Blinky\Blink	y.c\main\dlyVal==0x320',
Expression: \\Blinky\Blinky.c\main\dlyVal==0x320	Read Write
Count:	Size:
Command:	□ 1 I I Dytes
	-

×

TIP: If you have a hard time getting the Watchpoint to trigger on 0x320, change the value to something you see in **dlyVal** as you change the pot to make matching easier. Sometimes it is difficult to get the pot to match exactly the trigger value.

- 12. You can also use the static variable **leds** with values of 1, 2 or 4. Acquiring a trigger is easier with this variable.
- 13. There are other types of expressions you can enter and are detailed in the Help button in the Breakpoints window.
- 14. To repeat this exercise, rotate the pot and click on RUN two or three times to get past the true condition.
- 15. Delete the Watchpoint with Ctrl-B and Kill All.
- 16. Stop the program \bigotimes and exit Debug mode.

TIP: If you have trouble displaying these variables in the Watch window or with the Watchpoints, change them to global rather than static.

TIP: You cannot set Watchpoints on-the-fly while the program is running like you can with hardware breakpoints.

TIP: To edit a Watchpoint, double-click on it in the Breakpoints window and its information will be dropped down into the configuration area. Clicking on Define will create another Watchpoint with any modifications. You should delete the old one by highlighting it and click on Kill Selected or try the next TIP:

TIP: The checkbox beside the expression allows you to temporarily unselect or disable a Watchpoint without deleting it. when they are configured.

reakpoints		×
Current Breakpoints:		
▼ 00: (A readwrite 0x20000004 len=4), 'counter == 0x5',		
		F
	- Access	
Expression:	Read	Write
Count: 1 -	Size:	
	1 🛋	E Bytes
Command:	,	🗖 Objects
Define Kill Selected Kill All Clo	se	Help

Hardware Breakpoints are also visible in this window

TIP: Raw addresses can also be entered into the Watchpoint. An example is: *((unsigned long *)0x20000004)

14) MTB: Micro Trace Buffer:

The NXP LPC800 processor contains an instruction trace called MTB. The trace buffer resides in the LPC800 internal RAM. The trace frames are stored here. The size of this buffer is adjustable in the file DBG_MTB.ini.

MTB works with any Keil ULINK and LPC-Link 2. It does not work with a J-Link <yet>.

- 1. Stop the processor \bigotimes and exit debug mode.
- 2. Open the project C:\Keil\ARM\Boards\Embedded Artists\LPCXpresso LPC800\Blinky_MTB\Blinky.uvproj.
- 3. If you are using ULINK2 or ULINLK-ME, select it in the Target selector:
- Click on Rebuild and program the Flash with Load
- 5. Enter Debug mode.
- 6. Select the Instruction Trace window by selecting View/Trace/Instruction Trace or by using this icon: Click on the small arrow beside the upper left icon to open this up:
- 7. A window similar to next one below will be visible: Size it accordingly. Note the instructions displayed. This is a record of the last number of instructions (2204 in this case) executed by the processor.

Disassembly Instruction Trace

- 8. Click on RUN 💷 and STOP. 🤒
- Note each time the program is cycled this way, different trace information is displayed.
- 10. Click on RUN
- 11. Set a breakpoint in the while(1) loop. The program will soon stop at this point.
- 12. Note C source is now displayed as shown in the Instruction Trace shown below:
- 13. Click and hold on the Instruction Trace tab and pull this window out into the open so you can see the Disassembly window at the same time.
- 14. Double-click on any line to go to this spot in the Blinky.c and Disassembly window as shown below: I clicked on 0x00622.
- 15. Remove the breakpoint.
- 16. Single step 🙌 a few times and note the Instruction Trace window is updated with the latest instruction executed.

 Image: Trace Exceptions

 Event Counters

 Instruction Trace

 Records

Filter: Execution • # Address Opcode Instruction Source 2193 0x00000684 D1FB BNE 2194 0x0000067E STM r1!,{r0} 2195 0x00000680 1F12 SUBS r2,r2,#4 CMP 2196 0x00000682 2A00 r2 #0x00 2197 0x00000684 D1FB BNE 0x0000067E 2198 0x00000686 4770 BX Ir 0x000002DE 3410 ADDS r4,r4,#0x10 2199 r4,r6 2200 0x000002E0 42R4 CMP 0x000002D4 BCC 2201 0x000002E2 D3F7 2202 0x000002E4 F7FFFEF0 _main_after_scatterload (0x000000C8) BL.W 2203 0x000000C8 4800 LDR r0,[pc,#0];@0x000000CC 0x00000CA 4700 2204 BX rO

Instruction Tra	ace			τ×
Filter: Ex	ecution		T	<u> </u>
#	Address	Opcode	Instruction Source	
24865	0x00000630	4C06	LDR r4.[pc,#24];@0x0000064C	
24866	0x00000632	6060	STR r0,[r4,#0x04]	
24867	0x00000634	B290	UXTH r0,r2 75: retur	n(ladVal);
24868	0x00000636	BD10	POP {r4.pc} 76: }	
24869	0x000001DC	492C	LDR r1.[pc,#176];@0x00000290	
24870	0x000001DE	8008	STRH r0.[r1,#0x00]	
24871	0x000001E0	4608	MOV r0,r1 56: dly\	/al = dlyVal << 5;
24872	0x000001E2	8800	LDRH r0.[r0,#0x00]	
24873	0x000001E4	0540	LSLS r0,r0,#21	
24874	0x000001E6	0C00	LSRS r0,r0,#16	
24875	0x000001E8	8008	STRH r0.[r1,#0x00]	
24876	0x000001EA	482A	LDR r0.[pc,#168];@0x00000294 58: leds	s <<= 1;
				•
Disasseml	oly 🔜 Instruc	tion Trace		

How it Works: The trace frames are stored in a

portion of the LPC800 internal RAM. In this case, it is set to 2 Kb. As the program is run, this memory is over written with the most recent instructions. When the program is stopped, the trace frames currently in the

RAM are appended to the Trace Instruction window which is stored in the host computer.

It is important to configure the IRAM settings to prevent the compiler from using any MTB RAM. See the section **Reserve IRAM Memory for MTB** two pages from this one.

TIP: CMSIS-DAP adds the options to clear, save and search trace. ULINK2, ULINK-ME and LPC-Link 2 uses CMSIS-DAP mode. A LPC-Link 2 was not tested for this document <yet>.



15) Exploring the MTB:

The MTB provides a record of instructions executed. No CPU cycles are stolen to accomplish this. MTB is part of CoreSight debugging technology.

- 1. Make sure all breakpoints and watchpoints are removed. Enter Ctrl-B or Debug/Breakpoints and select Kill All.
- 2. Exit Debug mode if it is in this mode.
- 3. Re-enter Debug mode. ⁴ This has the effect of restarting your entire program and the processor from a fresh start. The program will run to main() as directed. The following Data Trace window (or similar) will display:

Note these items:

- 1. There are 1,333 instructions captured.
- 2. The last instruction executed was: BX r0 located at memory 0x00CA.
- 3. Register R0 (see the Registers window) equals 0x199. The least significant bit means this is a Thumb2 instruction. Subtract one from this to get the actual address. (199-1=198)
- 4. The PC equals this 0x198 which was set by the BX instruction and R0. The PC can also be viewed in the Register window.

struction	race			
Filter: E	xecution		•	
#	Address	Opcode	Instruction	Source
1318	0x00000668	D1FB	BNE 0x00000662	
1319	0x00000662	C101	STM r1!,{r0}	
1320	0x00000664	1F12	SUBS r2,r2,#4	
1321	0x00000666	2A00	CMP r2,#0x00	
1322	0x00000668	D1FB	BNE 0x00000662	
1323	0x00000662	C101	STM r1!,{r0}	
1324	0x00000664	1F12	SUBS r2,r2,#4	
1325	0x00000666	2A00	CMP r2,#0x00	
1326	0x00000668	D1FB	BNE 0x00000662	
1327	0x0000066A	4770	BX Ir	
1328	0x000002C2	3410	ADDS r4,r4,#0x10	
1329	0x000002C4	42B4	CMP r4,r6	
1330	0x000002C6	D3F7	BCC 0x000002B8	
1331	0x000002C8	F7FFFEFE	BL.Wmain_after_scatterload (0x000000C8)	
1332	0x000000C8	4800	LDR r0.[pc,#0];@0x000000CC	
1333	0x000000CA	4700	BX r0	

122

BL> 123: BLX R0 323: void SystemInit (void) {

333: LPC_IOCON->PIO0_8 &= ~(0x3 << 3):

328: | PC_SYSCON->SYSAHBC| KCTR| |= ((0x1 << 7) | (0x1 << 18));

- 5. Select the Disassembly window tab. The yellow arrow address 0x198 is the next instruction to be executed which is MOVS r5,#0x01.
- 6. Double-click on the BX instruction in the Instruction Trace window. (or perhaps the next one LSLS if visible)
- 7. Examine this instruction in the Disassembly window. Note there is no clue this BX would result in the next instruction being MOVS or what caused the jump to the MOVS at 198. The MTB has faithfully recorded that it was.

Examine the Start of Program:

8. Scroll to the top of Trace Data. We want to look at the very first instruction executed but this has been over-written.

BSS

0x00000006 4780 0x00000456 B508

0x00000458 4820 3080 6800 493D

0x0000045A 0x0000045C

0x0000045E

0x00000460 4308 0x00000462 492A 0x00000464 3180

0x00000466 6008

0x0000048E 4388 0x00000470 4939

483B 6B80 2118 0x00000468 0x0000046A

0000D4 4804

Filter: Execution

9 10

11 12 13

14

٠

LDR r0.fpc #

BLX r0 PUSH {r3,lr}

STR

LDR LDR

BICS r0.r0.r1

ADDS r0,r0,#0x80 LDR r0,[r0,#0x00] LDR LDR

LDB r0 Inc #1761 @0x0000050C

ORRS r0.r0.r1 LDR r1.[pc.#168] ; @0x0000050C ADDS r1.r1.#0x80

LDR r1.[pc.#228] ; @0x00000558

r0.fr1.#0x001

r0.[pc,#236] r0.[r0,#0x38]

MOVS r1.#0x18

r1.[pc.#244] ; @0x00000554

@0x00000558

- 9. Double-click on the first instruction.
- 10. Set a breakpoint at this instruction in the Disassembly window.
- 11. Exit and re-enter Debug mode. The processor will stop at the breakpoint. Scroll to the top of the Instruction Trace window. This window displays:
- 12. Note the first instruction is at 0xD4 and is a LDR instruction. This is the very first instruction executed after RESET.
- 13. Open a Memory window and enter address 0x0. Right click and select Unsigned Long. See the window below:

0x0000046C

0x0000046E 4388

- 14. Note 0X0 is the Initial Stack Pointer (0x1000 0208).
- 15. 0x4 is the initial PC and is $0x100 \ 00D5$. Subtract one to get the actual initial PC = 0xD4 as is displayed in the Trace.
- 16. Click on Step (F11) a few times and see more instructions executed and recorded in the Trace Data window.
- 17. Remove breakpoints: Enter Ctrl-B and select Kill All and Close.

TIP: If Run to main() is not set, (Target Options and select Debug tab) no instructions will be executed when Debug mode is entered. The PC will be at the first instruction. You can Step (F11) or RUN from this point and the Trace Data window will update as the instructions are executed.

TIP: These addresses will probably be different with different compiler options. The addresses shown were obtained with MDK 4.70 defaults.

Memory 1			Ļ	×
Address: 0			ц.	
0x00000000:	10000208	00000D5	000000E5	
0x0000000C:	00000DD	00000000	00000000	
0.000000181	0000000	FFFF <u>FR61</u>	00000000	_
🚰 Call Stack + L	ocals 🛛 🛄 Mer	mory 1		

16) Configuring the MTB Micro Trace Buffer:

You will learn how to configure MTB for your own project. We will configure the first Blinky to have MTB implemented.

- 1. Exit Debug mode. Q Open Project/Open Project.
- 2. Open the file: C:\Keil\ARM\Boards\Embedded Artists\LPCXpresso LPC800\Blinky\Blinky.uvproj.

Select the Initialization File:

- 1. Click on Target Options and select the Debug tab.
- The file DBG MTB.ini needs to be inserted in the Initialization box as shown here: 2.
- 3. This file is located in C:\Keil\ARM\Boards\Embedded Artists\LPCXpresso LPC800\Blinky MTB. You might want to copy this file into C:\Keil\ARM\Boards\Embedded Artists\LPCXpresso LPC800\Blinky.
- You can use the Browse icon to locate and select it. 4.
- 5. When selected, click on Edit... to open it up.

Reserve IRAM Memory for MTB (so the user program will not use it)

- 6. Click on the Target tab.
- 7. Note the IRAM1 has a length of 0x1000 (4096 bytes). We must reserve some (0x800 = 2Kb) for MTB.
- 8. Set IRAM1 to a length of 0x800 a shown here:
- 9. Click on OK to return. The compiler will not allocate RAM > 0x800.

TIP: If this is not correct, a Hard Fault often is the consequence.

TIP: The SWV trace (in Debug tab, Settings and then the Trace tab) must not be enabled. If it is, MTB will not be activated.

10. Click on Rebuild and program the Flash with Load

Configure the Initialization File:

- 11. DBG MTB.ini will now be displayed with the other source files because Step 5 opened it.
- 12. It is a small function that writes to several registers to enable and configure MTB.
- 13. Click on the Configuration Wizard tab at the bottom. The next screen opens: Expand the menu items.
- 14. You can enable/disable and change the size of the trace buffer here. Leave all entries at the default at this time.

Alert: "Stop Target when buffer is full" does not work on the first LPC800 silicon. This will be fixed in the next release. Everything else works fine.

TIP: After modifying this file (is indicated by an asterisk beside its name), select File/Save All. Exit and re-enter Debug mode to re-run this file. The Reset icon

will clear the trace and sometimes re-run the ini file.

Run The Program:

- Enter Debug mode 🛰 1.
- Open the Instruction Trace window using the icon: or by selecting View/Trace/Instruction Trace. 2.
- Click on RUN 🖳. When you stop 🥙 the program, the Trace Data window will be updated. 3.
- Select Stop Trace when Buffer is full in the Dbg MTB.ini file. 4.
- Select File/Save All. Exit and re-enter Debug mode ⁴ to clear the trace and reset the target. 5.
- Note the Trace Gap as shown below. This indicates 6. where trace collection has been stopped.
- Step (F11) a few times and note the Trace displays 7. these traces as shown here:
- When you are done, change Dbg MTB.ini back to its 8. default settings.
- 9. Remove any breakpoints and Watchpoints. (Ctrl-B)

Expand All Collapse All	Help Show Grid					
Option	Value					
-Trace: MTB (Micro Trace Buffer)	v					
Buffer Size	2kB					
Buffer Position	0×10000800					
Stop Trace when buffer is full						
Stop Target when buffer is full						
Trace: MTB (Micro Trace Buffer)						
•	Þ					
Text Editor Configuration Wizard	_					

Instructi	on Tra	ace						4	×
Filter	: Ex	ecution		•]				-
#		Address	Opcode	Instructi	on	Source			
170	2	0x0000067E	C101	STM	r1!.{r0}	1			
170	3	0x00000680	1F12	SUBS	r2,r2,#4				
170	4	0x00000682	2A00	CMP	r2,#0x00				
170	5	0x00000684	D1FB	BNE	0x0000067E				
170	6	0x0000067E	C101	STM	r1!.{r0}				
170	7	TRACE GAP							
170	8	0x00000198	F7FFFFCE	BL.W	LED_Init (0x00000138)	47: LED_Init();	/* LED Initialization	*/	
170	9	0x00000138	B500	PUSH	{r}	24: void LED_Init(void) {			-
C Disassembly 🗉 Instruction Trace									

Load Application at Startup

Blinky_MTB\Dbg_MTB.ini

IRAM1:

IRAM2:

0x10000000

Initialization File:

Run to main()

0x800

Edit.

Г

Г

- ×

17) Trace "In the Weeds" Example

Perhaps the most useful use of trace is to show how your program got to an unexpected place. This can happen when normal program flow is disturbed by some error and it goes "into the weeds". A record of all the instructions executed prior to this usually catastrophic error are recorded to the limits of the trace buffer size. In a Cortex-M processor, when a program goes haywire, it normally ends up at one of the error exception vectors. This minimizes damage to your system and preserves it for subsequent debugging. The MTB trace has a record of how the program got there.

- 1. Use your MTB enabled Blinky from the previous page. We need MTB enabled for this exercise.
- 2. Stop the program of running. Be in Debug mode.
- 3. In the file startup_LPC8xxx.s, set a breakpoint at the Hard Fault exception as shown here:
- 4. You can also find the Hard Fault near address 0x00DC in the disassembly window.
- 5. This stops the program when a Hard Fault occurs.
- 6. Setting this breakpoint stops the program when a Hard Fault occurs. Otherwise a Hard fault handler, which is normally a branch to itself, will fill up the trace buffer with this one instruction. This is not useful.

136

137

138

139

- 7. Click on RUN 🖳 After a few seconds, click on STOP. 🥸 The Instruction Trace window will be full.
- 8. In order to crash the program, we can execute a POP out of order, an illegal instruction or use a bogus lr (R14) value.
- 9. Usually the program will stop in the Delay function in Blinky.c. Start and stop the program until it does.
- 10. At 0x182 will see a BX lr instruction as shown below: When executed, BX places the contents of lr (R14) in the PC.
- 11. In the Register window, double-click on the R14 lr register value and enter 0x0. This is sure to cause a hard fault.
- 12. Click on RUN
- 13. The program will immediately go to the Hard Fault vector and the breakpoint will stop execution at the B *before it is executed*.
- 14. The Call Stack window will correctly show the program is in the Hard Fault Handler.
- 15. The Instruction Trace now shows the last number of instructions executed. You can clearly see the instructions executed before the fateful BX as shown here:
- 16. Click on Step (F11) a few times and the B at the HardFault_Handler will be executed and displayed as shown on the bottom window: Remember, a CoreSight breakpoint does not execute the instruction it is set to.
- 17. Note the Call Stack window displays the entry into the Hard fault vector as __asm_0xD4.
- 18. Remove the breakpoint.
- 19. Exit Debug mode. 🔍

TIP: MTB can be used to more easily solve difficult program flow problems that often require much investigation. Trace can sometimes be the only way to find the cause of these problems.

32: void Delay	(uint32_t	; dlyTicks) {	
33: uint32_t	curTicks;		
34:			
0x00000182 4770	BX	lr	
35: curTicks	= msTicks	3;	
0x00000184 4A39	LDR	r2,[pc,#228] ; @0x0000026C	
0x00000186 6811	LDR	r1,[r2,#0x00]	
36: while ((m	sTicks -	<pre>curTicks) < dlyTicks) { NOP();</pre>	}
0x00000188 E000	в	0x0000018C	
0x0000018A BF00	NOP		
	T		
Disassembly Instruction	Trace		

LED.c Blinky.c Dbg_MTB.ini

PROC

EXPORT HardFault_Handler

.

[WEAK]

HardFault_Handler\

Filter: Fr	ace				ф X •			
#	Address	Opcode		Source				
5901	0x0000018A	BF00	LDR r2.[pc.#220] ; @0x00000268	1	_			
5902	0x0000018C	4A37	LDR r2,[r2,#0x00]					
5903	0x0000018E	6812	SUBS r2,r2,r1					
5904	0x00000190	1A52	CMP r2,r0					
5905	0x00000192	4282	BCC 0x00000188					
5906	0x00000194	D3F9	BX Ir <					
5907	0x00000196	4770	MOVS r5,#0x01	37: }				
					•			
Disassem	Disassembly 🖳 Instruction Trace							

Filter: E	xecution		_		
#	Address	Opcode	Instruction	Source	
5904	0x00000190	1A52	CMP r2,r0		
5905	0x00000192	4282	BCC 0x00000188		
5906	0x00000194	D3F9	BX Ir		
5907	0x00000196	4770	MOVS r5,#0x01	37: }	
5908	0x000000DC	E7FE	B HardFault_Handler (0x00000DC)	139:	В
5909	0x000000DC	E7FE	B HardFault_Handler (0x00000DC)	139:	В
5910	0x000000DC	E7FE	B HardFault_Handler (0x000000DC)	139:	В

18) RTX_Blinky Example Program with Keil RTX RTOS:

Keil provides RTX, a full feature RTOS. RTX is included as part of Keil MDK including source. It can have up to 255 tasks and no royalty payments are required. This example explores the RTX RTOS project. MDK will work with any RTOS. An RTOS is just a set of C functions that gets compiled with your project.

RTX comes with a BSD license and source code. See https://www.keil.com/demo/eval/rtx.htm and www.arm.com/cmsis

- 1. With µVision in Edit mode (not in debug mode): Select Project/Open Project.
- 2. Open the file C:\Keil\ARM\Boards\Embedded Artists\LPCXpresso LPC800\RTX_Blinky\Blinky.uvproj.
- 3. Compile the source files by clicking on the Rebuild icon.
- 4. To program the Flash manually, click on the Load icon. 4. A progress bar will be at the bottom left.
- 5. Enter the Debug mode by clicking on the debug icon and click on the RUN icon.
- 6. The red led will blink in accordance with the two tasks running.
- Click on STOP ³

The Configuration Wizard for RTX:

- 1. Click on the RTX_Conf_CM.c source file tab as shown below on the left. You can open it with File/Open if needed.
- 2. Click on the Configuration Wizard tab at the bottom and your view will change to the Configuration Wizard.
- 3. Open up the individual directories to show the various configuration items available.
- 4. See how easy it is to modify these settings here as opposed to finding and changing entries in the source code.
- 5. Changing an attribute in one tab changes it in the other automatically. You should save a modified window.
- 6. You can create Configuration Wizards in any source file with the scripting language as used in the Text Editor.
- This scripting language is shown below in the Text Editor as comments starting such as a </h>
 or <i>.
 See <u>www.keil.com/support/docs/2735.htm</u> for instructions.
- 8. The µVision System Viewer windows are created in a similar fashion. Select View/System Viewer.

/ 🗈	RTX_Conf_CM.c 🗸 🗸
081	#ifndef OS_TICK
082	#define OS_TICK 10000
083	#endif
084	
085	//
086	// <e>Round-Robin Task switching</e>
087	//
088	// <i> Enable Round-Robin Task switching</i>
089	<pre>#ifndef OS_ROBIN</pre>
090	#define OS_ROBIN 1
091	#endif -
092	
093	// <o>Round-Robin Timeout [ticks] <1-1</o>
094	// <i> Define how long a task will exe</i>
095	// <i> Default: 5</i>
096	#ifndef OS_ROBINTOUT
097	#define OS_ROBINTOUT 5
141	#endif
Text Edito	r 🖉 Configuration Wizard 🖌

Blinky.c 🔝 RTX_Conf_CM.c	▼ ×
Expand All Collapse All	Help
Option	Value
₽ Task Definitions	
- Number of concurrent running tasks	7
- Number of tasks with user-provided stack	0
- Task stack size [bytes]	200
- Check for the stack overflow	V
- Run in privileged mode	
- Number of user timers	0
- SysTick Timer Configuration	
- Timer clock value [Hz]	72000000
Timer tick value [us]	10000
e Round-Robin Task switching	✓
- Round-Robin Timeout [ticks]	5
\ Text Editor \ Configuration Wizard /	

Text Editor: Source Code

Configuration Wizard

19) RTX Kernel Awareness using RTX Viewer

Users often want to know the number of the current operating task and the status of the other tasks. This information is usually stored in a structure or memory area by the RTOS. Keil provides a Task Aware window for RTX. Other RTOS companies also provide awareness plug-ins for μ Vision or use an external program for this purpose.

- 1. Run RTX_Blinky by clicking on the Run icon.
- 2. Open Debug/OS Support and select RTX Tasks and System and the window below opens up. You might have to grab the window and move it into the center of the screen. Note these values are updating in real-time using the same technology as used in the Watch and Memory windows.
- 3. Select View and select Periodic Window Update if these values do not change:

You will not have to stop the program to view this data. No CPU cycles are used. Your program runs at full speed. No instrumentation code needs to be inserted into your source. µVision automatically configures this when RTX is present.

RTX Tasks and System										
-	•									
Property	Value	2								
	Item			Valu	ie					
	Timer	Number:		0						
	Tick T	imer:		10.0	00 mSec					
	Roun	d Robin Timeout:		50.0	00 mSec					
	Stack	Size:		200						
	Tasks	with User-provided Stack	c	0						
	Stack Overflow Check:			Yes						
	Task Usage:			Available: 3, Used: 2						
	User Timers:			Available: 0, Used: 0						
Ė Tasks	ID	Name	Priority	1	State	Delay	Event Value	Event Mask	Stack Load	
	255	os_idle_demon	0		Running				0%	
	3	ledOff	1		Wait_DLY	7			32%	
	2	ledOn	1		Wait_DLY	7			32%	

Demonstrating States:

Blinky.c contains two tasks. Task 1 (phaseA) is shown below:

- 1. The gray areas opposite the line numbers indicate there is valid assembly code located here.
- Set a breakpoint on one of these in Task 1 ledOn in Blinky.c as shown: Do not set it on the for (;;;) at line 23.
- 3. Set a second breakpoint in Task 2 ledOff at a similar location.
- 4. Click on RUN

- Blinky.c RTX_Conf_CM.c Abstract.txt startup_LPC8xx.s LED.c × . 22 task void ledOn (void) { 23 for (;;) 24 25 LED On (0); /* Turn LED On os evt set (0x0001, t ledOff); /* send event to task 'ledoff' 26 delay 50 clock ticks os dly wait (50); 27 3 28 29 30 31 Task 2 'ledOff': switches the LED off 32 1
- 5. When the program stops, this information will be updated in the RTX Tasks window. The Task running when the program stopped will be indicated with a "Running" state. Most of the time the CPU is executing the os_idle_demon.
- 6. Each time you click on RUN, the program will stop in the next task and this will be updated in RTX Tasks.
- 7. Remove the breakpoints and close the RTX Tasks window.
- 8. Exit Debug mode. 🔍

20) DSP SINE example using ARM CMSIS-DSP Libraries:

ARM CMSIS-DSP libraries are offered for Cortex-M0, Cortex-M0+, Cortex-M3 and Cortex-M4 processors. DSP libraries are provided in MDK in C:\Keil\ARM\CMSIS. README.txt describes the location of various CMSIS components. For more information please see <u>www.arm.com/cmsis</u>, and <u>forums.arm.com</u>.

CMSIS is an acronym for Cortex Microcontroller Software Interface Standard.

This example creates a *sine* wave, then creates a second to act as *noise*, which are then added together (*disturbed*), and then the noise is filtered out (*filtered*). The waveform values in each step are displayed in the Watch 1 window.

This example incorporates Keil RTX RTOS. RTX is available free with a BSD type license. Source code is provided.

- 1. To obtain this example file, go to <u>www.keil.com/appnotes/docs/apnt_237.asp</u> and copy it into C:\Keil\ARM\Boards\Embedded Artists\LPCXpresso LPC800\. A \DSP directory will be created.
- 2. Stop μ Vision \bigotimes and exit Debug mode \bigotimes if necessary.
- 3. Open the project file sine: C:\Keil\ARM\Boards\Embedded Artists\LPCXpresso LPC800\DSP\sine.uvproj
- 4. Build the files. If There will be no errors or warnings.
- 5. Program the LPC800 flash by clicking on the Load icon: Progress will be indicated in the Output Window.
- 6. Enter Debug mode by clicking on the Debug icon. Select OK if the Evaluation Mode box appears.
- 7. Click on the RUN icon.
- Select View/Watch Windows and select Watch 1. This will open as shown below with the four global variables sine, noise, disturbed and filtered declared in DirtyFilter.c being updated in real-time: These variables were entered into Watch 1 previously.

Watch 1		X
Name	Value	Туре
🔷 🔮 sine	0xCF0E	short
🗝 🔗 noise	0x0800	short
🚽 🔗 disturbed	0xD336	short
🔷 🖗 filtered	0xC34F	short
<enter expression=""></enter>		

9. The NXP Cortex-M3 and Cortex-M4 processors have Serial Wire Viewer (SWV). The four waveforms will be displayed in the Logic Analyzer as shown below. The NXP LPC800 does not have SWV so this screen is shown for reference only. You can see the four global variables graphed:



RTX Tasks and System Kernel Awareness with the DSP Demonstration:

- 1. Click on RUN to start the program if not already running.
- 2. Open Debug/OS Support and select RTX Tasks and System. A window similar to below opens up. You might have to drag it into the center of the screen to see it more easily.
- 3. Note this window does not update: nearly all the processor time is spent in the idle daemon. The processor spends relatively little time in each task in this example. These timings are easily adjusted with RTX.
- 4. Set a breakpoint in each one of the four tasks in DirtyFilter.c by clicking in the left margin on a grey area. These tasks are sine_gen (near line 78), noise_gen (near line 96), disturb_gen (near line 116) and filter_tsk (near line 136). Do not set your breakpoints on the while(1) statement. It will not stop here.
- 5. The program will stop here and the Task window will be updated accordingly. Here, I set a breakpoint in the disturb_gen task:

Disassembly 📮 🗙									
Γ	0x00000366	4F3D	LDR	r7, [pc, #244]	;	@0x0000045C 🔺			
	0x00000368	46BC	MOV	r12,r7					
	0x000036A	DF00	SVC	0x00					
	116: disturbed = sine + noise;								
e	0x0000036C	483D	LDR	r0,[pc,#244]	;	@0x00000464			
	0x0000036E	8800	LDRH	r0,[r0,#0x00]					
	0x00000370	4940	LDR	r1,[pc,#256]	;	@0x00000474			
	0x00000372	8809	LDRH	r1,[r1,#0x00]					
	0x00000374	1840	ADDS	r0,r0,r1		•			
┛									
Disassembly III Instruction Trace									
	DirtyFilter.c is sine_generator.c is system_LPC8xx.c = ×								
Г	110 -								
	111ta	sk void di	isturb_ger	n(void)					
	112 🖂 {								
	113 while(1)								
	115	os_evt_was	Lt_and (0x0	JUUI, UXFFFF);					
	<pre>116 disturbed = sine + noise;</pre>								
	<pre>117 os_evt_set(0x0001, filter_tsk_tid);</pre>								
•	110 5 }								

- 6. Clearly, in the RTX Tasks and System window below, you can see that noise_gen was running when the breakpoint was activated. Os_idle_demon is Ready to run when noise_gen is finished and no other task is Ready.
- 7. Each time you click on RUN, a new Task will run. When it stops, the task that was running is indicated.

	KIA Tasks and	i bysten							· * ^
TIP: os_idle_demon has a Priority of	-		•						
0 which is the lowest priority	Property	Value							
possible. Everything other task has a	⊡System	Item			Value				
higher priority		Time	r Number:		0				
ingher priority.		Tick 1	îimer:		10.000 mSec				
8. Remove the breakpoints		Round Robin Timeout:			50.000 mSec				
(Ctrl-B)		Stack Size:		200					
(eur 2):		Tasks with User-provided Stack:		0					
9 Stop the processor 🥸		Stack Overflow Check:		Yes					
y. Stop the processor = .		Task Usage:		Available: 7, Used: 5					
10. Close the Tasks window.		User Timers: Available: 0, Used: 0							
11 Exit debug mode 🔍	Tasks	ID	Name	Priority	State	Delay	Event Value	Event Mask	Stack Load
11. Exit debug mode.		255	os_idle_demon	0	Ready				32%
		6	sync_tsk	1	Wait_DLY	1			32%
		5	filter_tsk	1	Wait_AND		0x0000	0x0001	32%
TIP: Recall this window uses the		4	disturb_gen	1	Running		0x0000	0x0001	0%
CoreSight DAP read and write		3	noise_gen	1	Wait_AND		0x0000	0x0001	32%
technology to undate this window and		2	sine_gen	1	Wait_AND		0x0000	0x0001	32%
does not steal CPU evalue									
uuus nuu suun uu u u u u u u us									

21) Configuring MTB for the DSP SINE example: (for reference)

It is easy to configure MTB for any project. Recall that MTB stores trace data in the processor internal RAM. It is important to properly configure the RAM so the user program will not overrun this memory. A Hard fault can happen. MTB is preconfigured in the example provided. This page describes all the pertinent settings.

This DSP example uses a great deal of the LPC800 internal RAM. When you build this project the Build Output window will display the amount of RAM needed. In this case it is 136 + 2888 = 3,024 (0xBD0) bytes. This needs to be allocated.



- MTB RAM: The LPC800 has 4 Kb (0x1000)
 RAM. This means we have at maximum only 4096-3024 = 1072 for MTB trace. We will round this to 1 Kb (1024 bytes). We are required to use RAM in certain boundaries.
- 2. User RAM: We can maximum use 4096-1024 = 3072 (0xC0) bytes.

Configure IRAM1:

- 3. Click on Target Options and select the target tab. Note the RAM size is set to 1000. We need to change this to the value the program needs. Change it to 0xC00 as shown in this window: MTB will use the rest.
- 4. The first available address for MTB will be $0x1000_{0000} + 0xC00 = 0x1000_{0}C00$.

TIP: If this is not set correctly, a Hard Fault is often the consequence.

Configure Dbg_MTB.ini:

- 5. Click on the Debug tab. Enter the file Dbg_MTB.ini from example Blinky_MTB. Click on the Edit button to make it visible. Click on OK to close Target Options. You might want to copy a version over to the \DSP directory.
- 6. Select the Configuration Wizard tab at the bottom of the Dbg_MTB source tab.
- 7. This window opens up: Expand the elements if necessary.
- 8. Note the Buffer Size is default to 2kB.
- 9. Open the Buffer Size select 1kB as shown:
- 10. Change the Buffer Position to 0x1000 0C00:
- 11. Select File/Save All.

These are all the steps necessary to properly configure the MTB.

- 12. Build the files $\overset{(1)}{=}$ and program the flash:
- 13. Enter Debug mode 🤐 and click on RUN 💷



14. Open the Instruction Trace window and the executed instructions will be displayed. Most will be the branch instruction to itself which is the os_idle_demon. Set a breakpoint in sine_generator.c near line 64 for a more interesting display as shown: You can open any source file from the projects window.

This is the end of the exercises.

Next is how to create your own projects. Plus...

- A review of what trace is good for.
- Keil product and contact information.

NXP LPCXpresso Cortex-M0+ Lab with ARM[®] Keil™ MDK toolkit

	instruction in	ace				+ ^
ho	Filter: Ex	ecution		•		
.ne	#	Address	Opcode	Instruction	Source	
	1548	0x000001B4	2302	MVNS r3,r3		
	1549	0x000001B6	43DB	BX r3		
wn	1550	0x000001B8	4718	PUSH {r4,lr}		
	1551	0x0000033C	484C	DCD 0x484CF000	96: noise = sine_calc_sample_q15(&Noise_set) / 6;	
	1552	0x0000033E	F000F941	DCD 0xF000F941		
ace is	1553	0x000005C4	B51C	MOV r4,r0	60: {	
	1554	0x000005C6	4604	MOVS r1,#0x00		
	1555	0x000005C8	2100	STR r1,[sp,#0x04]	63: input = 0;	
	1556	0x000005CA	9101	MOVS r3,#0x01		
ontact	<u>I</u>		_			▁
	🕄 Disassem	bly 🔜 Instruc	ction Trace			

Copyright © 2013 ARM Ltd. All rights reserved www.keil.com

•	on-chip IRAM1:	0x10000000	0xC00	
	IRAM2:			

22) Creating your own project from scratch: Using the Blinky source files:

All examples provided by Keil are pre-configured. All you have to do is compile them. You can use them as a starting point for your own projects. However, we will start this example project from the beginning to illustrate how easy this process is. We will use the existing source code files so you will not have to type them in. Once you have the new project configured; you can build, load and run a bare Blinky example. It has an empty main() function so it does not do much. However, the processor startup sequences are present and you can easily add your own source code and/or files. You can use this process to create any new project, including one using an RTOS.

Create a new project called Mytest:

- 1. With µVision running and not in debug mode, select Project/New µVision Project...
- 2. In the Create New Project that opens, go to the folder C:\Keil\ARM\Boards\Embedded Artists\LPCXpresso LPC800\.

Create a new folder and name your project:

- 3. Right click inside this window and create a new folder by selecting New/Folder. I named this new folder FAE.
- 4. Double-click on the newly created folder "FAE" to enter this folder. It will be empty.
- 5. Name your project in the File name: box. I called mine Mytest. You can choose your own name but you will have to
- keep track of it. This window is shown here:6. Click on Save.

Select your processor:

- 7. Select Device for "Target 1" opens up as shown below.
- 8. This is the Keil Device Database[®] which lists all the devices Keil supports. You can create your own if desired for processors not released yet.
- Locate the NXP directory, open it and select LPC812M101 (or the device you are using). Note the device features are displayed for your reference.
- 10. Note the large number of NXP parts that Keil supports.
- 11. Select OK.
 - µVision will configure itself to this device.

Select the startup file:

- 12. A window opens up asking if you want to insert the default LPC8xx.s startup file to your project. Click on "Yes". This will save you some good time.
- 13. In the Project Workspace in the upper left hand of μVision, expand the folders Target 1 and Source Group 1 by clicking on the "+" beside each folder.
- 14. We have now created a project called Mytest with the target hardware called Target 1 with one source assembly file startup_LPC8xx.s and using the LPC800 Cortex-M0+ processor.

TIP: You can create more target hardware configurations and easily select them. This can include multiple Target settings, simulation and RAM operations. See Projects/Manage/Components when you complete creating your own project.

Rename the Project names for convenience:

- 15. Click once on the name "Target 1" (or twice if not already highlighted) in the Project Workspace and rename Target 1 to something else. I chose LPC800 Flash. Press Enter to accept this change. Note the Target selector in the main μVision window also changes to LPC800 Flash.
- 16. Similarly, change Source Group 1 to Startup. This will add some consistency to your project with the Keil examples. You can name these or organize them differently to suit yourself.
- 17. Select File/Save All.





Copyright © 2013 ARM Ltd. All rights reserved www.keil.com

Select the source files and debug adapter:

1. Using MS Explore (right click on Windows Start icon), copy files Blinky.c and system_LPC8xx.c from C:\Keil\ARM\Boards\Embedded Artists\LPCXpresso LPC800\Blinky to the \FAE folder you created.

Source Files:

- In the Project Workspace in the upper left hand of μVision, right-click on "LPC800 Flash" and select "Add Group". Name this new group "Source Files" and press Enter. You can name it anything. There are no restrictions from Keil.
- 3. Right-click on "Source Files" and select Add files to Group "Source Files".
- 4. Select the file Blinky.c and click on Add (once!) and then Close.

System File:

- 5. Right-click on "Startup" and select Add files to Group "Source Files".
- 6. Select the file system_LPC8xx.c and click on Add (once!) and then Close.
- 7. Your Project window will look similar to the one shown here:

Select your Debug Adapter:

- By default the simulator is selected when you create a new μVision project. You probably want to change this to a USB adapter such as a ULINK2.
- 9. Select Target Options in ALT-F7 and select the Debug tab. Select ULINK/ME Cortex Debugger as shown below: If you are using another adapter such as CMSIS-DAP or ULINK*pro*, select the appropriate adapter from the pull-down list.
- 10. Select JTAG/SWD debugging (as opposed to selecting the Simulator) by checking the circle just to the left of the word "Use:" as shown in the window to the right:
- 11. Select the Utilities tab and select the appropriate debug adapter. To select the proper Flash algorithm, click on Settings: The LPC800 uses the algorithm named LPC8xx IAP 16kB Flash. Click on OK once.
- 12. Click on the Target tab and select MicroLIB for smaller programs. See <u>www.keil.com/appnotes/files/apnt202.pdf</u> for details.
- 13. Click on OK to close this window and return to the main μ Vision menu.

Modify Blinky.c

- 14. Double-click the file Blinky.c in the Project window to open it in the editing window or click on its tab if it is already open.
- 15. Delete everything in Blinky.c except the main () function to provide a basic platform to start with:

#include <lpc8xx.h> /*</lpc8xx.h>		
Main Program		+ /
<pre>int main (void) {</pre>		*/
<pre>while(1) { // your code goes here !</pre>	/* Loop forever	*/
}		

16. Select File/Save All

Compile and run the program:

- 17. Compile the source files by clicking on the Rebuild icon.
- 18. Program the LPC800 Flash by clicking on the Load icon: Progress will be indicated in the Output Window.
- 19. Enter Debug mode by clicking on the Debug icon.

NXP LPCXpresso Cortex-M0+ Lab with ARM[®] Keil™ MDK toolkit

- 20. Click on the RUN icon. 💷 Note: you stop the program with the STOP icon. 🤒
- 21. The program will run but since while(1) is empty it does not do very much. But it runs. You can set a breakpoint in the Disassembly window on the Branch to itself to demonstrating the program makes it this far.
- 22. You should be able to add your own source code to create a meaningful project.

This completes the exercise of creating your own project from scratch.

You can also configure a new RTX project from scratch using RTX_Blinky project.



Ŧ

Settinas

Linker Debug Utilities

ULINK2/ME Cortex Debugger

X

23) NXP LPC800 Cortex-M0+ Trace Summary:

Watch and Memory windows can see:

- Global variables.
- Static variables.
- Structures.
- Peripheral registers just read or write to them.
- Can't see local variables. (just make them global or static).
- Can't see DMA transfers DMA bypasses CPU and CoreSight and CPU by definition.

Serial Wire Viewer displays in various ways: : (Cortex-M0+ does not have SWV. Cortex-M3, Cortex-M4 does)

- PC Samples.
- Data reads and writes.
- Exception and interrupt events.
- CPU counters.
- Timestamps are provided.
- ITM Instrumentation Trace Macrocell: A *printf* feature.

Instruction Trace (MTB) is good for:

- Trace adds significant power to debugging efforts. Tells where the program has been.
- A recorded history of the program execution *in the order it happened*.
- Trace can often find nasty problems very quickly.
- Weeks or months can be replaced by minutes.
- Especially where the bug occurs a long time before the consequences are seen.
- Or where the state of the system disappears with a change in scope(s).

These are the types of problems that can be found with a quality trace:

- Pointer problems.
- Illegal instructions and data aborts (such as misaligned writes).
- Code overwrites writes to Flash, unexpected writes to peripheral registers (SFRs), a corrupted stack. *How did I get here ?*
- Out of bounds data. Uninitialized variables and arrays.
- Stack overflows. What causes the stack to grow bigger than it should ?
- Runaway programs: your program has gone off into the weeds and you need to know what instruction caused this. Is very tough to find these problems without a trace. ETM trace is best for this.
- Communication protocol and timing issues. System timing problems.

24) Useful Documents: See www.keil.com/NXP

- 1. The Definitive Guide to the ARM Cortex-M0 by Joseph Yiu.
- 2. The Definitive Guide to the ARM Cortex-M3 by Joseph Yiu. Search the web to obtain these books.
- 3. MDK-ARM Compiler Optimizations: Appnote 202: <u>www.keil.com/appnotes/files/apnt202.pdf</u>
- 4. **A handy list of resources is located at:** <u>http://www.arm.com/products/processors/cortex-m/index.php</u> Click on the Resources tab. Or search for "Cortex-M3" on <u>www.arm.com</u> and click on the Resources tab.
- 5. forums.arm.com and www.arm.com/cmsis
- 6. CAN: Controller Area Network lab with a LPC1700 Cortex-M3 processor: <u>www.keil.com/download/docs/385.asp</u>

25) Keil Products and Contact Information:

Keil Microcontroller Development Kit (MDK-ARM™)

- MDK-Lite (Evaluation version) \$0
- **NEW !!** MDK-ARM-CM[™] (for Cortex-M series processors only unlimited code limit) \$3,200
- MDK-Standard (unlimited compile and debug code and data size) \$4,895
- MDK-Professional (Includes Flash File, TCP/IP, CAN and USB driver libraries) \$9,995

For special promotional or quantity pricing and offers, contact Keil Sales.

USB-JTAG adapter (for Flash programming too)

- ULINK2 \$395 (ULINK2 and ME SWV only no ETM)
- ULINK-ME sold only with a board by Keil or OEM.
- ULINK*pro* \$1,250 Cortex-M*x* SWV & ETM trace.
 All ULINK products support MTB.

The Keil RTX RTOS is now provided under a Berkeley BSD type license. This makes it free.

All versions, including MDK-Lite, includes Keil RTX RTOS with source code !

Keil provides free DSP libraries for the Cortex-M0+, Cortex-M3 and Cortex-M4.

Call Keil Sales for details on current pricing, specials and quantity discounts. Sales can also provide advice about the various tools options available to you. They will help you find various labs and appnotes that are useful.

All products are available from stock.

All products include Technical Support for 1 year. This is easily renewed.

Call Keil Sales for special university pricing. Go to <u>www.arm.com</u> and search for university to view various programs and resources.

Keil supports many other NXP processors including ARM7[™] and ARM9[™] series processors. See the Keil Device Database[®] on <u>www.keil.com/dd</u> for the complete list of NXP support. This information is also included in MDK.

Note: USA prices. Contact sales.intl@keil.com for pricing in other countries.

Prices are for reference only and are subject to change without notice.

For Linux, Android and bare metal (no OS) support on Cortex-A processors, please see DS-5 www.arm.com/ds5.

For more information:

Keil Sales: In the USA: sales.us@keil.com or 800-348-8051. Outside the USA: sales.intl@keil.com

Keil Technical Support in USA: <u>support.us@keil.com</u> or 800-348-8051. Outside the USA: <u>support.intl@keil.com</u>.

International Distributors: www.keil.com/distis/ See www.embeddedsoftwarestore.com

For more NXP specific information: visit www.keil.com/NXP

CMSIS Version 3: www.arm.com/cmsis and forums.arm.com







